

## 1. (20 pts.) Problem 1

(a) Similar to binary search, start by examining  $A[\lfloor \frac{n}{2} \rfloor]$ .

- If  $A[\lfloor \frac{n}{2} \rfloor]$  is  $\lfloor \frac{n}{2} \rfloor$ , then we have a satisfactory index.
- If  $A[\lfloor \frac{n}{2} \rfloor] > \lfloor \frac{n}{2} \rfloor$ , then no element in the second half of the array can possibly satisfy the condition. To check this, note that all integers in the array are distinct, so each integer in the array is at least one greater than the previous element. Since  $A[\lfloor \frac{n}{2} \rfloor]$  is already greater than  $\lfloor \frac{n}{2} \rfloor$ , all the elements on its right are greater than their indices as well. Thus, the second half can be discarded in this case.
- if  $A[\lfloor \frac{n}{2} \rfloor] < \lfloor \frac{n}{2} \rfloor$ , then by the same logic no element in the first half of the array can satisfy the condition.

We discard the half of the array that cannot hold an answer and repeat the same check until the satisfactory index has been found or all the elements in the array have been discarded.

- (b) At each step we do a single comparison and discard at least half of the remaining array (or terminate), so the running time of this algorithm is given by the recurrence  $T(n) = T(n/2) + O(1)$  and hence  $T(n) = O(\log n)$  time by the master theorem.

## 2. (20 pts.) Problem 2

(a)

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}^2 = \begin{bmatrix} a^2 + bc & ab + bd \\ ca + dc & cb + d^2 \end{bmatrix} = \begin{bmatrix} a^2 + bc & b(a + d) \\ c(a + d) & bc + d^2 \end{bmatrix}$$

Hence, the 5 multiplications  $a^2, d^2, bc, b(a + d)$  and  $c(a + d)$  suffice to compute the square.

(b) Two possible answers:

1) We have:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^2 = \begin{bmatrix} A^2 + BC & AB + BD \\ CA + DC & CB + D^2 \end{bmatrix} \neq \begin{bmatrix} A^2 + BC & B(A + D) \\ C(A + D) & BC + D^2 \end{bmatrix}.$$

Note that matrices don't commute! This means  $BC \neq CB$  in general, so we cannot reuse that computation. Also, matrix multiplication only has left-distributivity and right-distributivity. As a result,  $B(A + D) = BA + BD \neq AB + BD$  and  $C(A + D) = CA + CD \neq CA + DC$ . Thus, we end up getting more than 5 subproblems, and the recurrence  $T(n) = 5T(n/2) + O(n^2)$  does not make sense.

- 2) Originally, our problem is to square a matrix. However, to compute  $\begin{bmatrix} A & B \\ C & D \end{bmatrix}^2$ , there is no way of making all the subproblems the same as our initial problem (squaring a matrix). So, this can't be claimed as a divide-and-conquer approach.

(c) Given two  $n \times n$  matrices  $X$  and  $Y$ , create the  $2n \times 2n$  matrix  $A$ :

$$A = \begin{bmatrix} 0 & X \\ Y & 0 \end{bmatrix}$$

Now, it suffices to compute  $A^2$  to obtain the result of the matrix multiplication, as its upper left block will contain  $XY$ :

$$A^2 = \begin{bmatrix} XY & 0 \\ 0 & YX \end{bmatrix}$$

Let  $S(n)$  be the time to square a  $n \times n$  matrix. The general matrix multiplication  $XY$  can be calculated in time  $S(2n)$ . Given that  $S(n) = O(n^c)$ ,  $S(2n) = O((2n)^c) = O(2^c n^c) = O(n^c)$  since  $c$  is a constant as well as  $2^c$  which can be ignored as a coefficient.

Note: matrix  $A$  in the solution is not unique. The other possibilities for matrix  $A$  are  $\begin{bmatrix} 0 & X \\ 0 & Y \end{bmatrix}$ ,  $\begin{bmatrix} X & Y \\ 0 & 0 \end{bmatrix}$ ,  $\begin{bmatrix} X^2 & XY \\ 0 & 0 \end{bmatrix}$ ,  $\begin{bmatrix} Y & 0 \\ X & 0 \end{bmatrix}$ ,  $\begin{bmatrix} Y^2 & 0 \\ XY & 0 \end{bmatrix}$ ,  $\begin{bmatrix} 0 & 0 \\ Y & X \end{bmatrix}$ ,  $\begin{bmatrix} 0 & 0 \\ XY & X^2 \end{bmatrix}$ , and  $\begin{bmatrix} 0 & Y \\ X & 0 \end{bmatrix}$  ( $A^2 = \begin{bmatrix} YX & 0 \\ 0 & XY \end{bmatrix}$ ).

### 3. (20 pts.) Problem 3

It is sufficient to show how to find  $n$  in time  $O(\log n)$ , as we can use binary search on the array  $A[0, \dots, n-1]$  to find any element  $x$  in time  $O(\log n)$ . To find  $n$ , query elements  $A[0], A[1], A[2], A[4], A[8], \dots, A[2^i], \dots$ , until you find the first element  $A[2^k]$  such that  $A[2^k] = \infty$ . Then,  $2^{k-1} \leq n < 2^k < 2n$ . We can then do binary search on  $A[2^{k-1}, \dots, 2^k]$  to find the last non-infinite element of  $A$ . This takes time  $O(\log(2^k - 2^{k-1})) = O(\log n)$ .

### 4. (20 pts.) Problem 4

We will assume  $k \leq n$ . If this is not the case, then for  $k' = 2n + 1 - k$ , we are trying to find the  $k'$ -th largest element, and  $k' \leq n$ . But finding the  $k$ -th largest element and finding the  $k$ -th smallest element are symmetric problems, so with some minor tweaks we can use the same algorithm. So for brevity, we will only give the answer for when  $k \leq n$ .

Since  $k \leq n$ , the  $k$ -th smallest element can't exist past index  $k$  of either array. So we cut off all but the first  $k$  elements of both arrays. Now, we just want to find the median of the union of the two arrays. To do this, we check the middle elements of both arrays, and compare them. If the median of the first list is smaller than the median of the second list, then the median can't be in the left half of the first list or the right half of the second list. If the median of the second list is smaller, the opposite is true. So we can cut these elements off, and then recurse on the resulting arrays.

#### Pseudocode

**procedure** TWOARRAYSELECTION( $a[1..n]$ ,  $b[1..n]$ , element rank  $k$ )

Set  $a := a[1, \dots, k]$ ,  $b := b[1, \dots, k]$ ,

Let  $\ell_1 = \lfloor k/2 \rfloor$  and  $\ell_2 = \lceil k/2 \rceil$

**while**  $a[\ell_1] \neq b[\ell_2]$  and  $k > 1$  **do**

**if**  $a[\ell_1] > b[\ell_2]$  **then**

    Set  $a := a[1, \dots, \ell_1]$ ;  $b := b[\ell_2 + 1, \dots, k]$ ;  $k := \ell_1$

    Let  $\ell_1 = \lfloor k/2 \rfloor$  and  $\ell_2 = \lceil k/2 \rceil$

**else**

    Set  $a := a[\ell_1 + 1, \dots, k]$ ;  $b := b[1, \dots, \ell_2]$ ;  $k := \ell_2$

    Let  $\ell_1 = \lfloor k/2 \rfloor$  and  $\ell_2 = \lceil k/2 \rceil$

```

    end if
  end while
  if  $k = 1$  then
    return  $\min(a[1], b[1])$ 
  else
    return  $a[\ell_1]$ 
  end if
end procedure

```

### Proof of correctness

Our algorithm starts off by comparing elements  $a[\ell_1]$  and  $b[\ell_2]$ . Suppose  $a[\ell_1] > b[\ell_2]$ .

Then, in the union of  $a$  and  $b$  there can be at most  $k - 2$  elements smaller than  $b[\ell_2]$ , i.e.  $a[1, \dots, \ell_1 - 1]$  and  $b[1, \dots, \ell_2 - 1]$ , and we must necessarily have  $s_k > b[\ell_2]$ . Similarly, all elements  $a[1, \dots, \ell_1]$  and  $b[1, \dots, \ell_2]$  will be smaller than  $a[\ell_1 + 1]$ ; but these are  $k$  elements, so we must have  $s_k < a[\ell_1 + 1]$ .

This shows that  $s_k$  must be contained in the union of the subarrays  $a[1, \dots, \ell_1]$  and  $b[\ell_2 + 1, \dots, k]$ . In particular, because we discarded  $\ell_2$  elements smaller than  $s_k$ ,  $s_k$  will be the  $\ell_1$ th smallest element in this union.

We can then find  $s_k$  by recursing on this smaller problem. The case for  $a[\ell_1] < b[\ell_2]$  is symmetric.

If we reach  $k = 1$  before  $a[\ell_1] = b[\ell_2]$ , we can cut the recursion a little short and just return the minimum element between  $a$  and  $b$ . You can make the algorithm work without this check, but it might get clunkier to think about the base cases.

Alternatively, if we reach a point where  $a[\ell_1] = b[\ell_2]$ , then there are exactly  $k$  greater elements, so we have  $s_k = a[\ell_1] = b[\ell_2]$ .

### Running time analysis

At every step we halve the number of elements we consider, so the algorithm will terminate in  $\log(k)$  recursive calls. Assuming the comparison takes constant time, the algorithm runs in time  $\Theta(\log k)$ . Note that this does not depend on  $n$ .

## 5. (20 pts.) Problem 5

The intuition for the divide-and-conquer solution is to split the vector  $v$  into a top half  $v_T$  with the first  $\lfloor n/2 \rfloor$  elements and a bottom half  $v_B$  with the rest of the elements. We'll similarly split the result of the multiplication  $(H_k v)_T$  and  $(H_k v)_B$ . Using this notation and the recursive nature of  $H_k$ , we can split  $H_k v$  as follows:

$$H_k v = \begin{bmatrix} (H_k v)_T \\ (H_k v)_B \end{bmatrix} = \begin{bmatrix} H_{k-1} v_T + H_{k-1} v_B \\ H_{k-1} v_T - H_{k-1} v_B \end{bmatrix} = \begin{bmatrix} H_{k-1} (v_T + v_B) \\ H_{k-1} (v_T - v_B) \end{bmatrix}$$

This shows that we can find  $H_k v$  by calculating  $v_T + v_B$  and  $v_T - v_B$  and using this algorithm to recursively multiply  $H_{k-1} (v_T + v_B)$  and  $H_{k-1} (v_T - v_B)$ . The running time of this algorithm is given by the recurrence relation  $T(n) = 2T(\frac{n}{2}) + O(n)$ , where the linear term is the time taken to perform the two sums. This has solution  $T(n) = O(n \log n)$  by the Master theorem.

# Rubric:

## Problem 1, 20 pts

(a) 15 points

6 points: provide an algorithm that achieves what we want

4 points: it's truly faster than the brute force algorithm

5 points: explain why it's correct

(b) 5 points

3 points: provide a running time analysis that makes sense

2 points: reach to a correct conclusion on the running time

Note:

- Some students might analyze the running time correctly, but the given algorithm is not faster than the brute force way. In this case, they'll get full points for part (b) but lose 4 points for part (a).
- Some other students might provide an algorithm that's indeed faster than the brute force approach, but their running time analysis has certain issues. In this case, they'll have the 4 points for part (a) second rubric but lose points for part (b).

## Problem 2, 20 pts

(a) 5 points

3 points: correctly expand the squaring of a  $2 \times 2$  matrix

2 points: correctly identify the 5 needed multiplications

(b) 6 points as long as the student provides either of the possible answers

1) 3 points: correctly expand the squaring of a matrix with 4 sub-matrices and point out the non-commutativity property

3 points: explain how the non-commutativity property introduces an issue for the statement

2) 3 points: point out the sub-problems of squaring a matrix are not necessarily squaring a matrix

3 points: point out it's no longer divide-and-conquer if the main problem and its sub-problems are different

(c) 9 points

3 points: the explanation of squaring a  $2n \times 2n$  matrix can also be done in  $O(n^c)$  time makes sense

2 points: specify a reasonable  $2n \times 2n$  matrix  $A$

4 points: explain how they can obtain the result of a general matrix multiplication from  $A^2$

## Problem 3, 20 pts

(a) 5 pts for pointing out that to find any element in a sorted array, the running time is  $O(\log n)$

(b) 15 pts for solution how to find the last non-infinite element in the array

## Problem 4, 20 pts

(a) 8 pts for designing correct divide and conquer algorithm, and clear description of their algorithm.

Note that correct algorithm should give us the  $k^{th}$  element in  $O(\log k)$ , however, if their runtime was  $O(\log n)$ , they will still get 7pts for this part.

- (b) 6 pts for showing the correctness of their algorithm. They need to show that why their algorithm outputs  $k_{th}$  element
- (c) 6 pts for analyzing the running time. Two ways of doing this are: 1) either showing their algorithm has  $\log k$  recursive calls, and each call takes  $O(1)$ , so the total runtime will be  $O(\log k)$ . 2) or they can write the recurrent relation ( $T(n) = T(\frac{n}{2}) + O(1)$ ), and derive runtime from it. In this case writing recurrence gets 4 pts, and deriving the final runtime gets 2 pts.

**Problem 5, 20 pts**

- (a) 5 pts for identifying that the problem can be split into top and bottom
- (b) 10 pts for splitting the multiplication into 2 multiplications of size  $n/2$
- (c) 5 pts for showing the running time is  $O(n \log n)$